VoCard: An Effective Approach to English Vocabulary Acquisition

Min-Ji Kim, Chang-Woo Kang, Young Joon Eo, Ye-Hyun Jo

Capstone Design Project Fall 2023

December 8, 2023

Abstract

This paper describes *VoCard*, an effective vocabulary learning mobile applications with AI. It presents the overall implementation process and results of the *VoCard*, including its architecture, design and the process for features. *Vocard* provides English words, meanings, example sentences, and illustrations corresponding to the example sentences. It offers feedback on user-generated sentences and generates suitable images for them. Additionally, it adjusts the learning content based on the user's learning and test results, facilitating efficient memorization of words.

Keywords— English vocabulary learning, language application, active learning

1 Introduction

Smartphones have become essential and effective tools in modern society, offering portability, rapid internet access, and versatile application utilization. Users can download and install applications based on their needs through application stores such as the Google Play Store and the App Store. Among the diverse range of applications available in the app stores, the vocabulary learning category consistently garners popularity. The feature of smartphone apps, always accessible and portable, proves immensely helpful for language learning that requires consistent practice and memorization. This is because it allows studying and memorizing anytime, anywhere.

Therefore, our team focused on the project "*VoCard*," emphasizing English vocabulary learning. As a result of analyzing existing English vocabulary applications, we identified three key issues: development costs, meaningful application of newly learned vocabulary, and a lack of review sessions. Based on the issues identified in existing English vocabulary memorization applications, we planned to create an application that minimizes development costs and maximizes memorization efficiency. Consequently, we developed the *VoCard*, application with the assistance of AI tools such as ChatGPT and DALL-E, to offer users an enhanced vocabulary acquisition experience.

VoCard is a Vocabulary Learning Application that helps people learn English words efficiently with the help of powerful AI tools. *VoCard* provides the following functionalities: it generates example sentences for English words and offers feedback

on sentences created by the user, utilizing ChatGPT. Additionally, it generates images corresponding to sentences for each word through DALL-E. Furthermore, the system presents a card to the user from the card stack, and the displayed word card is selected based on the score calculated from the user's learning and testing records. Therefore, the user can study words that match their proficiency level.

VoCard is an iOS application developed using SwiftUI. For the backend, Django, specifically Django Rest Framework, and Google Firebase were utilized. The server was hosted on an AWS EC2 Instance. Application design was conducted using Figma to produce sensible design schemas and prototypes.

2 Design

2.1 Overall Architecture

2.1.1 Service Design

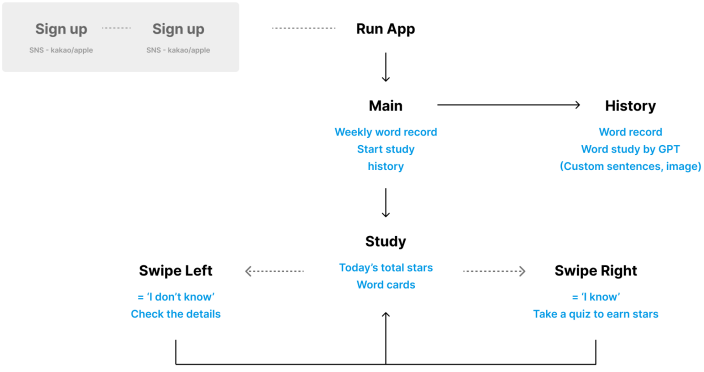

Figure 1: Service Architecture

In terms of service design within the Overall architecture subsection of the Design section, *VoCard* has been intricately crafted to provide a straightforward and intuitive user experience. The sign-in and sign-up functionalities for app execution aim to offer users a personalized learning experience, and the main screen allows easy access to weekly vocabulary learning records. This user-centric approach maximizes convenience for starting new learning sessions or reviewing previous learning records.

On the word memorization screen, users can visually track their daily learning progress through stars. The word cards are designed to be intuitive, effectively conveying detailed information. Swiping left on a word card signifies unfamiliarity, revealing detailed information such as the word's meaning, example sentences, and images. Additionally, swiping right indicates familiarity, allowing users to take quizzes to assess their understanding.

The word record screen organizes learned word cards by date and the number of stars achieved. This screen also incorporates GPT-powered learning, enabling users to

write example sentences for words, with the system generating contextually relevant images. This service design aims to provide users with an effective and enjoyable language learning experience by leveraging various interactive elements and personalized features.

2.1.2 System Architecture

Database Design Because we chose to use Google Firebase, we opted to utilize their cloud-based database system called Realtime Database. Its NoSQL structure allows development to be flexible in the face of evolving requirements and allows for an overall easily scalable system. The main purpose of our database is to prevent users from being restricted to the data that is on their local device. The database allows users to login to different devices and pick up where they left off seamlessly.

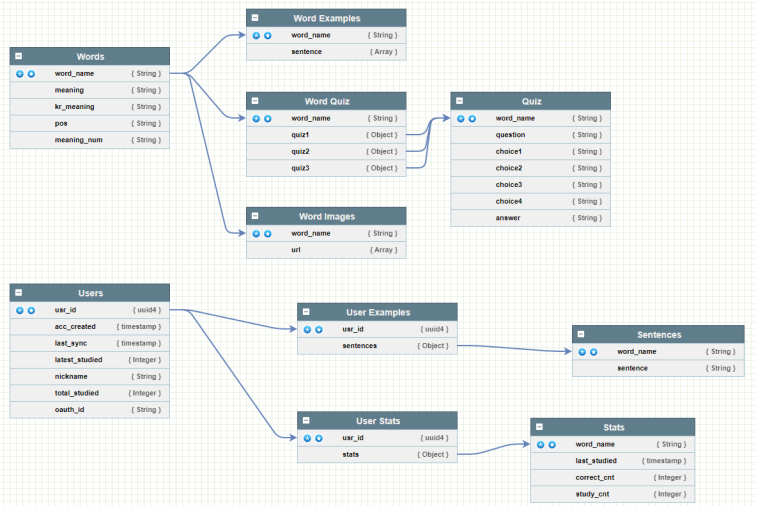

Figure 2: Firebase Database Schema (NoSQL)

Server Design Our Django application, created with Django-Rest-Framework for flexible RESTful API creation was hosted on an EC2 instance running NGINX in conjunction with Gunicorn. Supervisor for process control. NGINX was chosen due to its asynchronous and event-driven architecture. These features allow this web server to process many client requests simultaneously faster and with better performance than other web servers like Apache. Because NGINX is not a Python WSGI interpreter, we needed a WSGI-compatiable system in the middle of the NGINX web server and our Django application in order to run the application. Therefore, we chose to incorporate Guincorn, a WSGI server that is proven to work well with python web frameworks like Django. Gunicorn acts as the application web server that runs behind NGINX, the front-facing web server. Supervisor was utilized to make managing processes on our Linux EC2 instance easier.

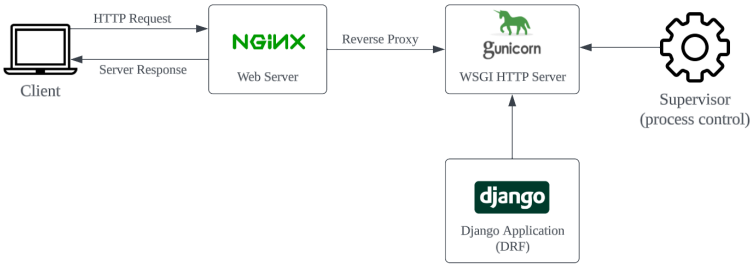

Figure 3: Server Design Diagram

2.2 Challenges

English-Korean Definition Extraction During development, we encountered difficulty in finding suitable Korean equivalents for English words. Due to the absence of Korean-English Dictionary APIs, web crawling emerged as the sole feasible method to programmatically obtain Korean word equivalents. However, this approach comes with its own set of challenges. For example, let's take the word 'bank,' which has two meanings. Daum and Naver are two major sources for crawling. Referring to the HTML of Daum's bank definition page, it becomes a challenge to programmatically find the correct equivalent without the definition being provided in advance. The Naver dictionary doesn't fare any better—it lacks the second definition equivalent in a reasonable location. We have two practical solutions at hand: manual review or simply utilizing ChatGPT as it provides an accurate equivalent for both meanings. However this raises a potential ethical concern of using primary vs tertiary sources. Would it not be better to refer to an official dictionary than using an AI trained on one? Ultimately, due to development constraints, we decided to utilize ChatGPT for Korean equivalents.

3 Implementation

Frontend Overview SwiftUI and XCode were utilized for the frontend, targeting iOS 17. The choice to target iOS 17 was primarily due to its support for SwiftData. SwiftData efficiently handles the transfer of schemas and instances from the Local DB on mobile devices to memory. Consequently, leveraging SwiftData was deemed essential for rapid development with a limited team. Efforts were made to align the Local DB as closely as possible with the backend schema. Data concerning user card reviews was also stored in the DB to facilitate easy creation and access. Moreover, operations that could potentially disrupt the app or detract from the user experience were executed asynchronously in threads separate from the UI, thereby enhancing user interaction.

The outcomes of collaboration with the server were stored in the Local DB as well. However, for development convenience, an option was implemented to store SwiftData instances only in memory. This approach resets all data upon the app's startup and closure. Originally, it was intended to retrieve all information related to the words on the cards from the server. Yet, to avoid unnecessary data transmission at app launch, the current beta version adopts a method of reading from a pre-stored JSON file and

storing it in the Local DB. All data flows were meticulously checked for nil values to minimize any risk of the app crashing.

Backend Overview The backend was created using Django and Django-Rest-Framework, a powerful and flexible toolkit for building RESTful APIs in Django. One of the main features of Django-Rest-Framework is the use of serializers that allows complex data such as querysets and model instances to be converted to native Python datatypes. These conversions can then be rendered in JSON, XML or other content types. This allows for simple and customizable class-based views with near plug-and-play levels of authentication and convenience. However, we could not utilize this feature since the database solution we are using, Firebase's Realtime Database, is not supported by Django's database engine. Therefore, we opted to use function-based views instead and manually link our database with our application. This implementation added significant development time since database manipulation needed to be done manually. One of the planned features for our application was the usage of OAuth from various social account providers such as Google, Facebook and Kakao. Although we were able to implement OAuth login through KakaoLogin, since our Alpha build does not use an online user system, this feature is not present in our Alpha build. Unit testing was conducted with our custom user model due to the importance of verifying user data integrity and consistency.

The API endpoints that feature default list information were designed as an alternative way of fetching or changing data in the database. Excluding special circumstances, these endpoints should not be called. Therefore, only the endpoints that dealt with user specific data and the OpenAI API integrations were designed to be called actively by our application. Of course, since our Alpha build does not user specific data manipulation, as of now, only the two endpoints that integrated OpenAI API are actively called by our application.

For image storage, we decided to use Firebase's Storage functionality. We stored images of words into folders since one word can have multiple meanings that we decide to incorporate into the default word list. These images are accessed using their URL. This URL is stored in the appropriate word entry in the database.

In order to document the API, we chose to use Swagger, a powerful suite of API developer tools. This solution provides a user interface to an API which allows for easy understanding of the API. Upon initial setup, Swagger automatically creates the user interface based on all of the installed apps in Django's settings.py. In most cases, the initial setup correctly gets the information for all methods related to a view, but since we were using function-based views, additional tweaking needed to be done. This fix was easily done by using Swagger's decorator to extend the schema and add the relevant information to the API endpoints and their methods. We utilized Swagger's extensive interactivity tools to allow our Frontend team to test the two endpoints that utilized the OpenAI APIs. This addition allowed our Frontend team to fully understand the format of request required and the type of response that would be returned to them without having to run into any errors during actual development.

3.1 Learning System

The study page displays word cards to the user, allowing them to swipe left or right. Swiping left indicates that the user doesn't know the word, while swiping right means that the user knows the word. If the user doesn't know the word (swipe left), they can view the meaning, associated image, and example sentence of the word. Conversely, if they know the word (swipe right), a quiz is provided to test their knowledge. The

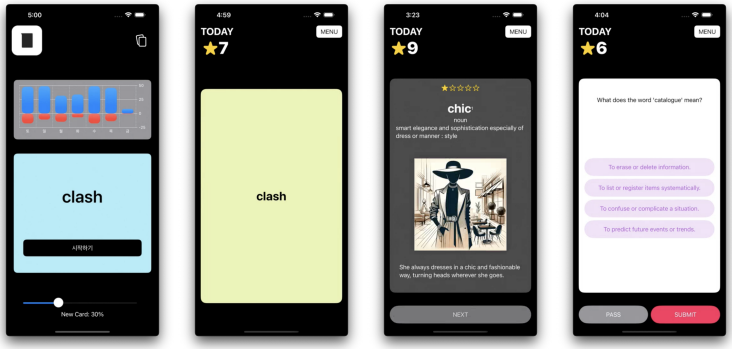

Figure 4: Learning System Workflow

stars shown in the attached screen image represent a scoring system for the user, and further details are explained in the following paragraph.

3.1.1 Study System in Detail

Our application applies the concepts of both The Ebbinghaus Forgetting Curve and spaced repetition to maximize user learning efficiency. In order to achieve this harmony, we devised a proprietary study system manipulates the probabilities of a card being drawn from a card deck. The system provides the user with a card from a card stack, or deck and the chance of a certain card being shown depends on an internal probability score. There are two categories of cards, "new cards" and "seen cards." For a "new card," or a card that the user has never interacted with before, the probability setting is set by the system to a default value. Users may change this probability setting in the application's settings page. The probability of a "seen card" being shown combines the principles of the Forgetting Curve and spaced repetition. Figure 5 depicts the piecewise function used for determining the probability of a "seen card" (represented as "SeenCard"). The first rule of this function addresses situations where a card is new or has never been reviewed before. If the recent review date of a card does not exist, that means that the card has never been interacted with before. The second case of this function addresses the way our system chooses a "seen card."

$$P(\text{card}_i) = \begin{cases} \frac{1}{\text{number of cards with null recent review date}} & \text{if recent review date of card}_i = \text{null} \\ \frac{\text{Review Elapsed Time}(\text{SeenCard}_i)}{\text{Required Review Interval}(\text{SeenCard}i)} \times \frac{1}{\sum_{k=1}^{n} \frac{\text{Review Elapsed Time}(\text{SeenCard}_k)}{\text{Required Review Interval}(\text{SeenCard}_k)}} & \text{otherwise} \end{cases}$$

Figure 5: Study System Piecewise Function

User Statistics For further explanation, it seems prudent to first explain how user statistic information on certain words is stored. Each user will have statistical information on words that they have encountered. Each word statistics entry will have the following fields shown in Figure 6; A timestamp indicating when that word was last studied, and a count of the number of times the user has correctly solved that

word's quiz. This number is not a cumulative count on the number of times a specific word's quiz has been solved, but rather the number of times that word's quiz has been successfully and successively solved. This counter would reset to 1 if a user gets two quizzes in a row correct but fails the third one.

Word (word_name)	w_1	w_2	...	w_n
Last studied timestamp (last_studied)	t_1	t_2	...	t_n
Number of Times Relevant Quiz Solved in Succession (correct_cnt)	c_1	c_2	...	c_n

Figure 6: User Statistic Information Based on a Word

Figure 7 shows the relationship between the recently introduced successive count and the study interval. This study interval value is derived from the insights of the Ebbinghaus Forgetting Curve. The study interval values reference commonly used spaced repetition intervals. For our case, we opted to use intervals of 30 minutes, 12 hours, 24 hours, 3 days, 1 week, and 1 month. If a user successfully solves the relevant quiz 6 or more times, it gets interpreted as the user fully knowing the word. Therefore, if a word's "c"value is high, then it should appear at a relatively later time compared to words that have a lower "c"value.

Number of Times Relevant Quiz Solved in Succession (c)	0	1	2	3	4	5	6
interval(c): Study Interval (hour)	½	12	24	24 x 3	24 x 7	24 x 30	inf

Figure 7: Relationship Between the Number of Successful Quizzes in Succession and Study Interval

Therefore, the probability of a Seen card being shown is the time since the last review of a Seen card divided by the review interval of the Seen card. This value is multiplied by 1 divided by the summation of this probability for all Seen cards. the probability value of a certain Seen Card is calculated through this piecewise function.

3.1.2 Frontend

In essence, cards are fundamentally categorized into two types: 1. Cards that have never been reviewed, characterized by possessing a ReviewResult. 2. Cards that have been previously reviewed, lacking a ReviewResult.

When users draw cards, they will encounter one of these two types of cards. Through the Slider on the MainPage, users have the option to select the probability of drawing cards that have never been seen before.

The probability of each card appearing in every card-drawing scenario is determined by two key factors: 1. The current star count. 2. The time elapsed since the last review.

The decision not to store cards in a max heap based on their weights is grounded in the consideration of computational efficiency. Storing cards in a max heap in the local database, according to their weights, was initially deemed a strategy to significantly reduce computational operations. However, since the "time elapsed since the last review" continually changes, storing it in a max heap would still necessitate recomputation and heap reconstruction at every card-drawing instance. Consequently, this approach was deemed impractical and devoid of meaningful computational benefits.

3.1.3 Backend

The Learning System utilizes a predefined list of words that contain meaning, Korean equivalent meaning, part of speech, quiz, image and example sentence. In the current implementation of our Alpha build, the application reads from a JSON file to get the default word list information without checking the database for any changes. Our production-ready build would refer to the database we have on Realtime Database and sync any changes in that database to the file on the user's local storage. Additionally, the planned scope of development was to track user changes and update the database accordingly. Although work has been done to implement this feature, since our Alpha build only runs locally, our application does not utilize the feature. Instead, it saves all information into its local database.

3.2 Review System

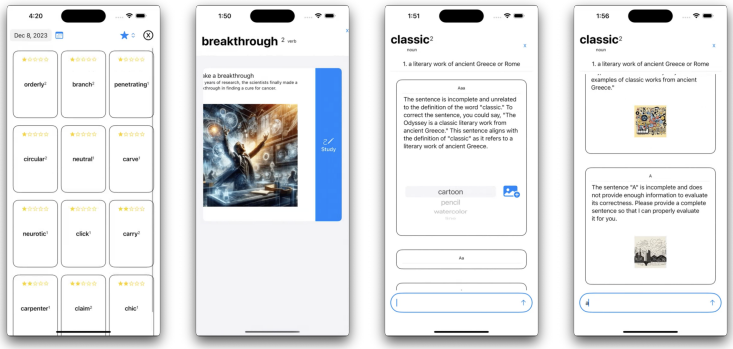

Figure 8: Review System Workflow

This page provides users with access to a comprehensive list of previously interacted cards. Clicking on a specific card opens its detailed view. Upon swiping right within the card's detailed view, users are directed to the user sentence input page, where they can input sentences for evaluation by ChatGPT. Users can generate images for their inputted sentences, choosing from a predefined list of styles. Notably, users have the flexibility to input multiple sentences consecutively without waiting for a server response, allowing them to generate images for different sentences without any required waiting time.

Figure 9: Pre Sentence Evaluation

3.2.1 Frontend

The following process occurs when a user creates a custom sentence. When the send button is pressed, the following code is executed. An asynchronous code is executed to send a request to the server to evaluate the given sentence. Afterwards, the result of this asynchronous code is received in the main thread and the UI update is carried out in the main thread. By doing this, the UI update is not interrupted, and asynchronous operations are executed in a different thread, with only the result being received. This allows multiple requests to be sent and received at once.

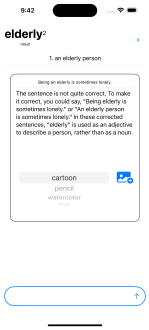

Figure 10: Post Sentence Evaluation

An appropriate evaluation was found for the sentence 'Being elderly is sometimes lonely'.

3.2.2 Backend

Sentence Evaluation The Review System utilizes the two API endpoints created for ChatGPT and Dall-E integration. These two endpoints are POST-based endpoints requiring a specific request for each endpoint. The endpoint for user sentence evaluation `/ai/eval/` requires `word` for the word name, `dfn` for the word's definition and `sentence` for the user's inputted sentence. The prompt uses these three arguments to evaluate the correctness of `sentence`. The `seed` parameter is used to enforce consistent

output from ChatGPT. The model used is `gpt-3.5-turbo-1106`. Per our testing, we concluded that that ChatGPT 3.5 is more than capable enough to provide accurate sentence correctness evaluation and thus have decided to incorporate this model into our API. Upon success, OpenAI API returns its documented return body for chat completions. Our server reads the response and only responds with the text generated by ChatGPT to the frontend's request.

Image Generation Another feature of our Review System is providing visual feedback of user generated sentences. After receiving evaluation from ChatGPT, regardless of whether the sentence was deemed correct or not, users can generate an image of their inputted sentence. For image generation, we initially decided to use Dall-E 2, but with the advent its more powerful successor, Dall-E 3, we changed to using the updated model despite the increased costs. The endpoint for image generation, `/ai/img/`, requires `style` for the image style and `sentence` for the user's inputted sentence. Upon sucess, OpenAI API returns its documented return body for image generations. Our server reads the response and only responds with the image URL to the frontend's request. Currently, we do not separately store this image onto our storage solution or server and thus the link is only valid for one hour[1].

4 Evaluation

Our project set out to demonstrate a system that is at minimum equivalent but preferably better than other existing English vocabulary memorization applications on the market with significantly lower development and upkeep costs. To provide a better service for less overall cost, we utilized ChatGPT, Dall-E and a cost-effective AWS EC2 instance. From a cost perspective, it seems clear that our application is much more cost-effective than other existing vocabulary learning applications.

4.1 Cost

One listing on a part-time service platform by the developers of 암기고래 show that they incur costs of at least 1000 Won per illustration. This cost is leaps and bounds more expensive than utilizing Dall-E 3, the most expensive and powerful image generation model OpenAI has to offer. Currently, the images we have for the words in our default word list are a mix of Dall-E 2 and Dall-E 3 generated images. This occurrence was due to Dall-E 3 being released for API access in late November of this year.

The same developers have recently posted a listing on Albamon, a popular Korean part-time service platform, that is looking for someone to create example sentences and provide verb conjugations for certain words. The pay they listed is ₩10,500/hour and it seems that they are looking for eight part-timers[3]. It goes without saying that using a tool like ChatGPT is a far more cost-effective tool procure such content. For our use case, we utilize ChatGPT 3.5 to provide evaluation and feedback on user generated example sentences. This is an example of a prompt that could be sent to OpenAI's chat completion endpoint for a ChatGPT response.

```
You are a teacher evaluating the correctness of a student's
    example sentence. If the example sentence is wrong, provide
    details on how to make the sentence correct. Evaluate the
    correctness of the sentence based on the definition of the
```

Model	Quality	Resolution	Price
DALL·E 3	Standard	1024×1024	$0.040 / image
	Standard	1024×1792, 1792×1024	$0.080 / image
DALL·E 3	HD	1024×1024	$0.080 / image
	HD	1024×1792, 1792×1024	$0.120 / image
DALL·E 2		1024×1024	$0.020 / image
		512×512	$0.018 / image
		256×256	$0.016 / image

Figure 11: Dall-E API Cost as of December 8, 2023[2]

```
word. Word: organization. Definition: characterized by
complete conformity to the standards and requirements of an
organization. Sentence: The employee's work was highly
organized, as it demonstrated complete conformity to the
standards and requirements of the organization.
```

This prompt total 88 tokens. Referencing ChatGPT 3.5 API pricing in Figure 12, one can see that the input cost would be

$$1 \cdot 10^{-6} \cdot 88 = 8.8 \cdot 10^{-5}$$

Model	Input	Output
gpt-4-1106-preview	$0.01 / 1K tokens	$0.03 / 1K tokens
gpt-4-1106-vision-preview	$0.01 / 1K tokens	$0.03 / 1K tokens
Model	**Input**	**Output**
gpt-4	$0.03 / 1K tokens	$0.06 / 1K tokens
gpt-4-32k	$0.06 / 1K tokens	$0.12 / 1K tokens
Model	**Input**	**Output**
gpt-3.5-turbo-1106	$0.0010 / 1K tokens	$0.0020 / 1K tokens
gpt-3.5-turbo-instruct	$0.0015 / 1K tokens	$0.0020 / 1K tokens

Figure 12: Various ChatGPT Models and their API Cost[2]

4.2 Results

We implemented all major features that were essential in providing the experience we were looking to deliver. The frontend and backend are linked through our server hosted on AWS EC2. For further iterations into a production-ready state, there are

a couple of major milestones that must be accomplished. First, user authentication and management needs to be implemented so that users are not confined to their local device. This implementation would allow us to use the infrastructure that we developed on Firebase to sync user data. Long term user testing should be conducted to test the effectiveness of our Study and Review System. User feedback will be crucial to further improve our application.

5 Limitations and Discussions

5.1 Dealing with Multiple Meanings

Our team received feedback on how to handle multiple meanings for each word. We has decided to assign one meaning to each word card for words with multiple meanings and to differentiate the words through numbering. However, we realized that attempting to include all definitions of a specific word would be unrealistic. For instance, the verb "run" has varied meanings. "Run" can mean engaging in physical activity by swiftly moving or racing, as exemplified in the sentence: "I like to run in the park every morning." Another sense of "run" involves operating or managing a business, machine, or organization, as seen in the example: "He decided to run his own business." Similarly, "run" can denote maintaining a particular state or managing a household, as demonstrated in the sentence: "She runs the household efficiently." In the context of devices or systems, "run" refers to their functioning or operation, as seen in: "The computer program is set to run automatically." Moreover, "run" can describe the flow of a liquid in a specific direction, exemplified by: "The river runs through the city." Lastly, in a musical context, "run" signifies playing an instrument or performing a piece, as illustrated by: "The musician will run through the song before the concert." Therefore, to address this limitation, we have chosen to deal with only two definitions widely recognized in common usage. The criteria for common usage was defined by consulting ChatGPT.

5.2 Choosing the Right Meanings

As mentioned in the previous subsection, we intended to select the two most commonly used meanings in order to deal with words that have multiple meanings. However, determining the criteria for "commonly used" definitions and establishing a universal standard for such definitions presented a challenge. In light of this recognition of complexity, and considering the constraints of the project timeline and scope, we chose to prioritize the goal of "efficient English word memorization learning." Due to these limitations, we decided not to delve extensively into the realm of sociolinguistics. Instead, we concluded that seeking guidance from ChatGPT can offer a straightforward solution to address this issue.

6 Related Work

6.1 Related Service: 암기고래

The application most closely aligned with our concept is a Korean app called "암기고래." This application offers pre-arranged word sets covering a range of categories, from elementary-level to more practical ones like business and travel. Each word set

includes a vocabulary list for 31 days, and users can freely learn the words assigned for that specific date. On a word-by-word basis, the application includes illustrations, phonetic pronunciations, and example sentences in both English and Korean. The app also offers a quiz, enabling users to assess various aspects of their word knowledge. This feature presents two types of quizzes—a spelling and definition pairing quiz—and users have the freedom to choose between them. However, it does not provide users with the opportunity to create example sentences using the newly introduced word; instead, it simply shows an example sentence.

7 Conclusion

In conclusion, we developed *VoCard* for a more efficient vocabulary learning experience compared to market alternatives. *VoCard* represents a significant step forward in vocabulary learning applications. By integrating cutting-edge AI technologies such as ChatGPT and DALL-E, along with the strategic use of spaced repetition intervals based on the Forgetting Curve, we aim to revolutionize the way users learn and retain vocabulary. The system not only allows users to create sentences, receive feedback, and generate visual aids but also presents learning materials in the form of word cards, aligning with the natural forgetting curve for optimized retention. With *VoCard*, we aspire to provide an engaging and effective platform that caters to diverse learning styles, fostering a deeper and more lasting understanding of new words for our users.

References

[1] OpenAI. *DALL·E API FAQ*. URL: https://help.openai.com/en/articles/6704941-dall-e-api-faq (visited on 12/10/2013).

[2] OpenAI. *Pricing*. URL: https://openai.com/pricing (visited on 12/10/2013).

[3] 알바몬. /재택또는출근근무/ 영어단어추가컨텐츠작성. URL: https://m.albamon.com/jobs/detail/99407192 (visited on 12/10/2013).